Динамическая память языка программирования. Динамические списки

Динамическая память языка программирования

1. Память, которую использует программа делится на три вида:
   Статическая память (static memory)
   a. хранит глобальные переменные и константы;
   b. размер определяется при компиляции.
2. Стек (stack)
   a. хранит локальные переменные, аргументы функций и промежуточные значения вычислений;
   b. размер определяется при запуске программы (обычно выделяется 4 Мб).
3. Куча (heap)
   a. динамически распределяемая память;
   b. ОС выделяет память по частям (по мере необходимости).

Динамически распределяемую память следует использовать в случае если заранее (на момент написания программы) не знаем сколько памяти нам понадобится (например, размер массива зависит от того, что введет пользователь во время работы программы) и при работе с большими объемами данных (например, массив из 1 000 000 int`ов не поместится на стеке).

До сих пор в программе использовались переменные и массивы, создаваемые компилятором языка. Однако при этом нерационально расходуется память, кроме этого некоторые задачи исключают использование структур данных фиксированного размера и требуют введения структур динамических, способных увеличивать или уменьшать свой размер уже в процессе работы программы. Основу таких структур составляют динамические переменные.

Динамическая переменная хранится в некоторой области ОП, не обозначенной именем, и обращение к ней производится через переменную-указатель.

В отличие от статических и автоматических данных, память под которые распределяется компилятором, динамически распределяемая память выделяется программой самостоятельно. Время жизни таких объектов также определяется программой. Память выделяется по мере необходимости и должна освобождаться, как только данные, содержащиеся в ней больше не нужны. Доступ к ней осуществляется при помощи указателей.

Функции создания динамических переменных и массивов объявлены в заголовочных файлах <alloc.h>, <stdlib.h>.

Функции выделения и освобождения памяти.

1. Функция void* malloc(размер) - выделяет в «куче» n байтов и возвращает адрес на 1-й байт, иначе возвращает 0(NULL), в реальности происходит обращение к операционной системе с просьбой выделить участок памяти необходимого размера и операционная система через менеджер памяти пытается данную просьбу удовлетворить и результатом работы данной функции будет либо адрес начала выделенной памяти, либо 0, что означает отсутствие выделения памяти. Необходимо применять преобразование типов.

```
void main(void){
        char *original="Исходная строка";
        char *copy;
        copy=(char*)malloc(strlen(original)+1);
        if(copy==NULL) {
                puts("Ошибка выделения памяти");
                exit(1);
        }
        strcpy(copy,original);
```

```
        cout<<copy<<endl; cout<<original<<endl;
        free(copy);
}
```

При выделении памяти она не очищается. Размер указывается в байтах.

2. Функция void* calloc(n,size type);

long newmas = (long*)calloc(100,sizeof(long));*
первый параметр - количество требуемых ячеек памяти;

второй параметр - размер каждой ячейки.

Функция calloc() обнуляет выделенную память.

```
 #define SIZE 128
 void main(void) {
char *str=(char *)calloc(1,SIZE);
        if(str==NULL) {
                puts("Ошибка выделения памяти");
                exit(1);
        }
else {
                cout<<"Введите строку";
                gets(str);
                cout<<str;
                free(s);
        }
}
```

3. void* realloc(void *, n) - изменяет размер ранее выделенного блока памяти. Если дополнительная память выделяется в новом месте оперативной памяти, то уже введенная информация переписывается в новое место.

ptr = realloc(nptr, size);
где nptr - указатель на ранее выделенный блок; size размер выделяемой памяти.

```
void main(void) {
        char *p1, *p2;
        puts(«Выделяем 128 байт»);
        p1=malloc(128);
        if(p1) {
                puts("Изменяем размер блока на 256 байтов");
                p2=realloc(p1,256); //блок теперь равен 256 байтам
        }
        if(p2)
                free(p2);
        else
                free(p1);
}
```

4. void free(void *ptr) - освобождение блока памяти, адресуемого указателем **ptr**.

Замечание: необходимо отметить, что при выделении памяти функцией **malloc(),** освободить можно только весь ранее запрошенный участок памяти, а при выделении памяти функцией **calloc(),** возможно освобождение памяти начиная с адреса большего чем начальный адрес памяти и до конца выделенного пространства.

Операции выделения и освобождения памяти.

В C++ есть свой механизм выделения и освобождения памяти — это функции **new** и **delete**.
Пример использования **new**:
*int * p = new int[1000000]; // выделение памяти под 1000000 int`ов*
Т.е. при использовании функции new не нужно приводить указатель и не нужно использовать sizeof().
Освобождение выделенной при помощи new памяти осуществляется посредством следующего вызова:
delete [] p;
Если требуется выделить память под один элемент, то можно использовать
*int * q = new int;*
или
*int * q = new int(10); // выделенный int проинциализируется значением 10*
в этом случае удаление будет выглядеть следующим образом:
delete q;

Замечание:
Выделять динамически небольшие кусочки памяти (например, под один элемент простого типа данных) не целесообразно по двум причинам:

1. При динамическом выделении памяти в ней помимо значения указанного типа будет храниться служебная информация ОС и C/C++. Таким образом потребуется гораздо больше памяти, чем при хранении необходимых данных на стеке.
2. Если в памяти хранить большое количество маленьких кусочков, то она будет сильно фрагментирована и большой массив данных может не поместиться.

Многомерные массивы.

new позволяет выделять только одномерные массивы, поэтому для работы с многомерными массивами необходимо воспринимать их как массив указателей на другие массивы.
Для примера рассмотрим задачу выделения динамической памяти под массив чисел размера n на m.

1ый способ

На первом шаге выделяется указатель на массив указателей, а на втором шаге, в цикле каждому указателю из массива выделяется массив чисел в памяти:

```
int ** a = new int*[n];
for (int i = 0; i != n; ++i)
a[i] = new int[m];
```

Однако, этот способ плох тем, что в нём требуется n+1 выделение памяти, а это достаточно дорогая по времени операция.

2ой способ

На первом шаге выделение массива указателей и массива чисел размером n на m. На втором шаге каждому указателю из массива ставится в соответствие строка в массиве чисел.

```
int ** a = new int*[n];
a[0] = new int[n*m];
for (int i = 1; i != n; ++i)
a[i] = a[0] + i*m;
```

В данном случае требуется всего 2 выделения памяти.

Для освобождения памяти требуется выполнить:

1ый способ:

```
for (int i = 0; i != n; ++i)
delete [] a[i];
delete [] a;
```

2ой способ:

```
delete [] a[0];
delete [] a;
```

Таким образом, второй способ опять же требует гораздо меньше вызовов функции delete [], чем первый.

Динамический список

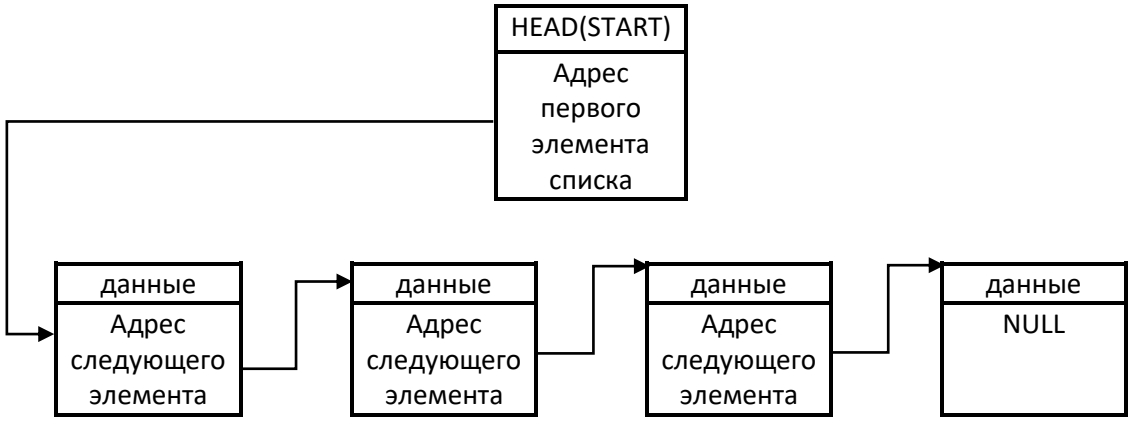
Связный список(список).

Структура данных, представляющая собой конечное множество упорядоченных элементов (узлов), связанных друг с другом посредством указателей, называется связным списком. Каждый элемент связного списка содержит поле с данными, а также указатель (ссылку) на следующий и/или предыдущий элемент. Эта структура позволяет эффективно выполнять операции добавления и удаления элементов для любой позиции в последовательности.

Причем это не потребует реорганизации структуры, которая бы потребовалась в массиве. Минусом связного списка, как и других структур типа «список», в сравнении его с массивом, является отсутствие возможности работать с данными в режиме произвольного доступа, т. е. список – структура последовательно доступа, в то время как массив – произвольного. Последний недостаток снижает эффективность ряда операций.

По типу связности выделяют односвязные, двусвязные, кольцевые и некоторые другие списки.

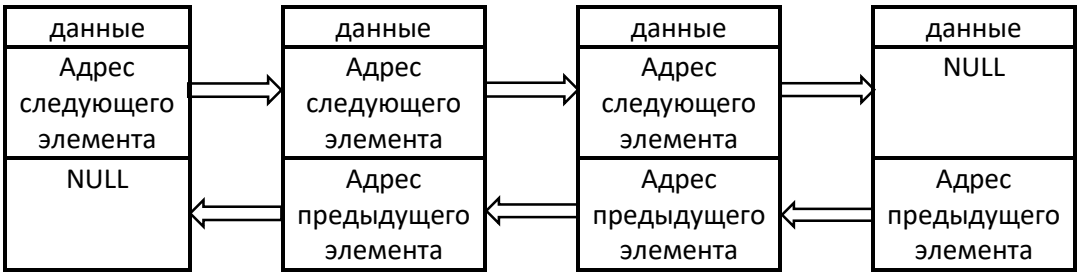
Каждый узел односвязного (однонаправленного связного) списка содержит указатель на следующий узел. Из одной точки можно попасть лишь в следующую точку, двигаясь тем самым в конец. Так получается своеобразный поток, текущий в одном направлении.



Односвязный список

На изображении каждый из блоков представляет элемент (узел) списка. Head – заголовочный элемент списка(самому списку не принадлежит), но содержит в себе адрес начального элемента списка. Признаком отсутствия указателя является поле, содержащее NULL.

Односвязный список не самый удобный тип связного списка, т. к. из одной точки можно попасть лишь в следующую точку, двигаясь тем самым в конец(либо в начало списка). Когда кроме указателя на следующий элемент есть указатель на предыдущий, то такой список называется двусвязным.

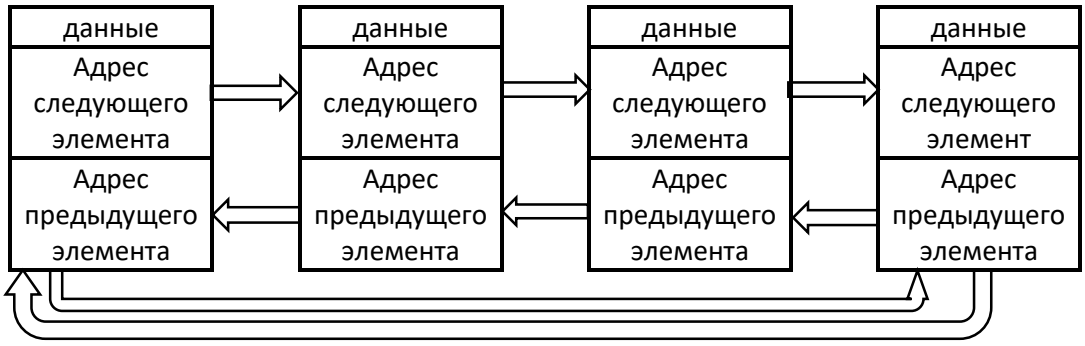


Двусвязный список

В двусвязном списке так же, как и в односвязном, присутствует элемент HEAD(START). Особенность двусвязного списка заключается в том, что каждый элемент имеет две ссылки: на следующий и на предыдущий элемент, позволяет двигаться как в его конец, так и в начало. Операции добавления и удаления здесь наиболее эффективны, чем в односвязном списке, поскольку всегда известны адреса тех элементов списка, указатели которых направлены на изменяемый элемент, но добавление и удаление элемента в двусвязном списке, требует изменения большого количества ссылок, чем этого потребовал бы односвязный список.

Возможность двигаться как вперед, так и назад полезна для выполнения некоторых операций, но дополнительные указатели требуют задействования большего количества памяти, чем таковой необходимо в односвязном списке.

Еще один вид связного списка – кольцевой список. В кольцевом односвязном списке последний элемент ссылается на первый. В случае двусвязного кольцевого списка – плюс к этому первый ссылается на последний. Таким образом, получается зацикленная структура.



Кольцевой список

Рассмотрим основные операции над связными списками.

Программная реализация списка.

На примере двусвязного списка, разберем принцип работы этой структуры данных. При реализации списка удобно использовать структуры.

Приведем пример разработанной программы. Комментарии в тексте программы дают представление о ее реализации.

// SPISOK.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

```
//
// реализация программы по построению двунаправленного
// кольцевого списка
// выполняются следующие действия
// ввод элемента
// удаление элемента
// замена элемента
// добавление элемента
// печать списка
// поиск элемента
// печать элемента
//получение адреса элемента списка по его id

#include "pch.h"
#include <iostream>
#include <windows.h>
#include <stdio.h>
using namespace std;

int fmenu(char **);          // объявление функции меню
// создаем структуру списка
struct Spis
{
        Spis *previous_item; //место для хранения адреса предыдущего элемента
        int d;               // значение элемента
        int id;              // идентификатор элемента списка создается автом начиная с 0
        Spis *next_item;     // место для хранения адреса последующего элемента
};
// окончание создания структуры списка


//-----------------------------------------------------
// создаем массив меню

char m1[] = { "1.ввод элемента                  :\n" };
char m2[] = { "2.удаление элемента по его id     :\n" };
char m3[] = { "3.замена элемента по его значению  :\n" };
char m4[] = { "4.добавление элемента             :\n" };
char m5[] = { "5.печать списка                   :\n" };
char m6[] = { "6.поиск элемента по его значению   :\n" };
```

```
char m7[] = { "7.печать элемента по его id       :\n" };
char m8[] = { "8.печать элемента по его значению  :\n" };
char m9[] = { "9.выход                          :\n" };
char *menu[] = {m1,m2,m3,m4,m5,m6,m7,m8,m9};
//--------------------печать----------------------------------
Spis *pStart,      // пер-ная. для хра-ния адреса начала списка
*pEnd,             // пер-ная. для хра-ния адреса последнего эл-та
*pCurrent,         // пер-ная. для хра-ния адреса текущего эл-та списка
*pTemp,            // пер-ная. для хра-ния промежуточного адреса эл-та списка
*pPrev;            // пер-ная. для хра-ния предыдущего адреса эл-та списка
int count0 = -1;

void input_item();             // функция ввода элемента списка
void del_item(int);            // функция удаления элемента списка по его id
void change_item(int);         // функция замены элемента списка по его значению
void add_item(int,int);        // функация добавления элемента в список 1 аргумент
                               // перед каким добавляем 2-ой что добавляем
void print_spis();             // печать всего списка
int find_item(int);            // поиск элемента списка по его значению возвращает его id
Spis* find_item(int,int fl);   // поиск элемента списка по его значению возвращает адрес
void print_item_id(int);       // печать элемента списка по его id
void print_item_value(int);    // печать элемента списка по его значению
void exit_programm();          // выход из программы
char z;
int main()
{

        SetConsoleCP(1251);      // подключаем русский язык
        SetConsoleOutputCP(1251);// на вывод и ввод
              // счетчик элементов списка -1 элем-тов нет
        Spis *pStart,          // пер-ная. для хра-ния адреса начала списка
             *pEnd,            // пер-ная. для хра-ния адреса последнего эл-та
             *pCurrent,        // пер-ная. для хра-ния адреса текущего эл-та списка
             *pTemp,           // пер-ная. для хра-ния промежуточного адреса эл-та списка
             *pPrev;           // пер-ная. для хра-ния предыдущего адреса эл-та списка
        for(;1;)
        switch (fmenu(menu))
        {
        case 1:input_item(); break;//вызов функции ввода элемента;
        case 2:int id; cout << "введи id для удаления:"; cin >> id;
                del_item(id); break;//вызов функции удаления элемента по его id;
        case 3:int value1; cout << "введи value для поиска и дальнейшей замены:";
                cin >> value1; change_item(value1); break;//вызов функции замены элемента по
его номеру в списке;
        case 4:int value3,value2; cout << "введи value для поиска:";
                 cin >> value3;
                      cout <<endl<< "введи value для добавления:";
                      cin >> value2;
```

```
                        add_item(value3,value2); break;//вызов функции добавление элемента на
место элемента
                                // с определенным значением;
                case 5:print_spis(); break;//вызов функции печать списка;
                case 6:int value,id_find; cout << "введи value для поиска:"; cin >> value;
                        id_find=find_item(value); cout << id_find<<endl; break;
                                //вызов функции поиск элемента по его значению;
                                // возвращает id найденного элемента
                case 7:int id0; cout << "введи id для поиска:"; cin >> id0;
                        print_item_id(id0); break;//вызов функции печать элемента по его номеру;
                case 8:int value0; cout << "введи value для поиска:"; cin >> value0;
                        print_item_value(value0); break;// печать элемента по его значению
                case 9:exit_programm();//вызов функции выход;
                default: {cout << "неверен пункт меню, повторите"; system("pause"); break;
                }
        }
}
void flush_stdin()
{
        cin.clear();
        while (cin.get() != '\n');
}
int fmenu(char *x[])
{
        int q;
        for (int i = 0; i < 9; i++)
                cout << x[i] ;  // вывод пунктов меню
        cin >> q;

        return q;               // возвращается пункт меню
}
void input_item()
{
        pCurrent = new Spis;
        cout << endl << "введите значение элемента:";
        cin >> pCurrent->d;
        if (count0 == -1) //не создавали ни одного элемента списка
        {
                count0 += 1;
                pCurrent->id = count0;
                pStart = pCurrent; // запомнили адрес начала списка
                pEnd = pCurrent;   // запомнили адрес последнего элемента списка
                pCurrent->previous_item = pCurrent;//  для 0-го элемента адрес предыдущего эл-та
                pCurrent->next_item = pCurrent;    // совпадает с адресом последующего эл-та
                pTemp = pCurrent;               // запомнили тек. адрес, потребуется при вводе
                                                                        //  следующего эл-
та
        }
```

```
		else
		{
			count0 += 1;
			pCurrent->id = count0;
			pEnd = pCurrent;	// запомнили адрес последнего элемента списка
			pCurrent->previous_item = pTemp;// запомнили в текущем адрес предыдущего эл-
та
			pCurrent->next_item = pTemp->next_item;// в поле след. текущего переписали
								// след. из предыщего
			//cin >> pCurrent->d;
			pStart->previous_item = pEnd;
			pTemp->next_item = pCurrent;
			pTemp = pCurrent;		// запомнили тек. адрес, потребуется при вводе
											//  следующего эл-
та
		}
	}
	void print_spis()
	{
		if (count0 < 0) { cout << "элементов в списке нет" << endl;
		system("pause"); return;
			}
		pCurrent = pStart;
		cout << pStart << endl;
		for (int i = 0; i <= count0; i++)
		{
			cout << pCurrent->previous_item << " : " << pCurrent->id << " : " <<
			pCurrent->d << " : " << pCurrent->next_item << endl;
			pCurrent = pCurrent->next_item;
	  }
		cout << pEnd << endl;
	}
	void print_item_id(int n)
	{
		int flag = 0;
		if (count0 < 0) {
			cout << "элементов в списке нет" << endl;
			system("pause"); return;
		}
		if (n > count0) {
			cout << "переданный номер больше махсимального в списке" << endl;
			system("pause"); return;
		}
		pCurrent = pStart;
		for (int i = 0; i <= count0; i++)
		{
			if (pCurrent->id == n) // совпадают ли id
			{
```

```
                flag = 1;
                cout << pCurrent->previous_item << " : " << pCurrent->id << " : " <<
                                pCurrent->d << " : " << pCurrent->next_item << endl;
                return;
                }
                else
                  pCurrent = pCurrent->next_item;    // получение адреса след. элемента
        }
        if (!flag) cout << "элемента с id=" << n << " нет в списке" << endl;
}
void print_item_value(int value)
{
        int d;
        int flag = 0;
        pCurrent = pStart;
        for (int i = 0; i <= count0; i++)
        {
                if (pCurrent->d == value)
                {
                        flag = 1;
                        cout << pCurrent->previous_item << " : " << pCurrent->id << " : " <<
                                pCurrent->d << " : " << pCurrent->next_item << endl;
                        cout << "искать далее?(1/0)"; cin >> d;
                        if (d==0)return;
                        pCurrent = pCurrent->next_item;
                }
                else
                        pCurrent = pCurrent->next_item;
        }
        if (!flag) cout << "элемента с зачением=" << value << " нет в списке" << endl;

}
void exit_programm() { exit(1); }
void del_item(int n) {
        int flag = 0;
        if (count0 < 0) {
                cout << "элементов в списке нет" << endl;
                system("pause"); return;
        }
        if (n > count0) {
                cout << "переданный номер больше махсимального в списке" << endl;
                system("pause"); return;
        }
        pCurrent = pStart;
        for (int i = 0; i <= count0; i++)
        {
                if (pCurrent->id == n)
                {
```

```
                flag = 1;
                // берем в найденном элементе адрес предыдущего и в поле следующего
                // предыдущего элемента записываем значение следующего из найденного
                pCurrent->previous_item->next_item = pCurrent->next_item;

                //берем в найденном элементе адрес последующего и в поле предыдущего
                // следущего элемента записываем значение предыдущего из найденного
                pCurrent->next_item->previous_item = pCurrent->previous_item;
                delete pCurrent;
                count0 -= 1;
                return;
            }
            else
                pCurrent = pCurrent->next_item;
        }
        if (!flag) cout << "элемента с id=" << n << " нет в списке" << endl;

}
int find_item(int value)
{
        int d;
        int flag = 0;
        pCurrent = pStart;
        for (int i = 0; i <= count0; i++)
        {
            if (pCurrent->d == value) // совпадают ли значения
            {
                flag = 1;
                cout << pCurrent->previous_item << " : " << pCurrent->id << " : " <<
                    pCurrent->d << " : " << pCurrent->next_item << endl;
                cout << "искать далее?(1/0)"; cin >> d;
                if (d == 0) return pCurrent->id;//возвращаем id найденного элемента
                pCurrent = pCurrent->next_item; flag = 0;
            }
            else
                pCurrent = pCurrent->next_item;
        }
        if (!flag) {
            cout << "элемента с зачением=" << value << " нет в списке" << endl;
            return -1;
        }
}
Spis* find_item(int value, int fl)
{
        int d;
        int flag = 0;
        pCurrent = pStart;
        for (int i = 0; i <= count0; i++)
```

```
	{
		if (pCurrent->d == value)
		{
			flag = 1;
			cout << pCurrent->previous_item << " : " << pCurrent->id << " : " <<
				pCurrent->d << " : " << pCurrent->next_item << endl;
			cout << "искать далее?(1/0)"; cin >> d;
			if (d == 0) return pCurrent;//возвращаем адрес найденного элемента
			pCurrent = pCurrent->next_item; flag = 0;
		}
		else
			pCurrent = pCurrent->next_item;
	}
	if (!flag) {
		cout << "элемента с зачением=" << value << " нет в списке" << endl;
		return nullptr;
	}

}
void change_item(int value1)
{
	Spis* x;
	int z;		// переменная для замены
	x = find_item(value1, 1);
		if (x == nullptr) return;
		cout << "введи значение для замены:"; cin >> z;
		cout << endl;
		x->d = z;
}
void add_item(int value3, int value2)
{
	Spis *pAdd,*pFind;
	pFind = find_item(value3, 1);//находим адрес искомого элемента
	if (pFind == nullptr) return;//если искомого нет то вываливаемся
	pAdd = new Spis;  // создаем новый элемент списка
	count0 += 1;// увеличиваем счетчик элементов и получаем новый id
	pAdd->d = value2;
	pAdd->id = count0;
	// заполнение полей
	// берем у найденного элемента(pFind) из поля предыдущий элемент(previous_item)
	// его адрес, и по этому адресу вытаскиваем из поле следующего элемента(previous_item)
	// адрес и заносим его в  поле (next)следующего добавляемого элемента по адресу pAdd
	pAdd->next_item = pFind->previous_item->next_item;
	// меняем на адрес pAdd поле next_item в
	//элементе pFind->previous_item
	pFind->previous_item->next_item = pAdd;
	// берем у найденного элемента(pFind) значение поля previous_item
	// и прописываем его в поле previous_item добавляемого элемента(pAdd)
```

```
        pAdd->previous_item = pFind->previous_item;
        // заменяем поле previous_item элемента pFind на значение pAdd
        pFind->previous_item = pAdd;
}
```